ARTUS Preliminary Development

by
Barton S. Wells
Frederick L. Beckner

Final Report on Contract DAAH01-99-C-R077
Option I
Cyberdynamics, Incorporated

January 7, 2000

U.S. Army Aviation and Missile Command
Redstone Arsenal, Alabama

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 7 Jan 00	3. REPORT TYPE AND DATES COVERED Final 2 Aug 99 - 30 Nov 99

4. TITLE AND SUBTITLE ARTUS Preliminary Development	5. FUNDING NUMBERS C DAAH01-99-C-R077
6. AUTHOR(S) Barton S. Wells Frederick L. Beckner	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Cyberdynamics Incorporated 1860 Embarcadero Road, Ste. 155 Palo Alto, CA 94303-3362	8. PERFORMING ORGANIZATION REPORT NUMBER CYB-0001
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Aviation & Missile Command AMSAM-AC-RD-A Redstone Arsenal, AL 35898-5200	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Work directed towards transitioning research conducted on the feasibility of an automatic rapid target updating system (ARTUS) to the implementation of system software is described. Implementation of one component of the ARTUS system, the ability to find features in terms of line intersections within infrared images, is performed. Methods of edge-detection, target filtering, line finding, segmentation of the lines, and segment intersection is implemented into a GUI-based software application. A new method of edge-detection is implemented using a Canny edge-detector combined with a second derivative gradient. An algorithm for filtering non-target data from target data in infrared images is created. Line finding is done using an algorithm taking the best lines from a histogram of all possible lines within an image. The lines are segmented using a technique searching for gaps in the lines, and then the intersections of these segments are found. This initial implementation of a portion of the ARTUS system is explained in this report.

14. SUBJECT TERMS Canny edge-detection, Gaussian, gradient, derivative, target filtering, line segments, line segment intersection, line breaks, histogram.	15. NUMBER OF PAGES 21
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

ARTUS Preliminary Development

Section 1: Overview

This report contains results of a transition from a feasibility study to the development and implementation of an automatic rapid target updating system (ARTUS) for use with Army missile guidance systems based on 2D infrared target images. These systems operate by matching the infrared image from a guidance sensor with predicted infrared images derived from a database of CAD models of a number of different possible targets. Such a system can be adversely affected by externally carried objects such as fuel tanks, supply crates, etc, not contained in the CAD model. The adverse effects of such objects could be minimized if there were a way to rapidly modify the CAD model to reflect the presence of such objects based on images obtained from reconnaissance sensors, in effect tailoring the CAD model to match specific targets.

Phase I of this contract dealt with studying the feasibility of developing such a software system, and developed a plan for the system and what algorithms would need to be developed to create software that could carry out these intended tasks. See the Phase I Final Report, on file at Cyberdynamics, Inc, and at the U.S. Army's Redstone Arsenal site in Alabama.

Phase I Option, which this report summarizes, began the task of implementing the plans of Phase I. A major portion of the ARTUS system would be to identify features of the target in both the infrared image and the CAD model. The most common feature is the intersection of lines, and that is what the Phase I Option work focused on.

Finding intersections of lines in both the infrared images and the rendered CAD models involves determining where there exists edges that can be extracted from the image, determining what lines are created by these edges and which points belong to which lines, reducing the lines to line segments, and determining which segments intersect within the image. Once the intersections are found, what type of intersection that is found needs to be determined. That is, how many line segments meet at the point of intersection? And, for each line segment, does the line segment cross the point or end at the point of intersection?

Work on the location of line intersection features for the ARTUS system was performed during the Phase I option. Many methods were investigated, and a successful method to find line intersections was found and implemented into software.

Section 2: Phase I Option Work

The following is a list of the work accomplished during the Phase I Option. In this phase we:

1. Experimented with "nearest-neighbor" techniques of combining adjoining points to form lines. This technique, only partially successful on simple images, resulted in too many and too short of lines, as well as creating a very difficult data-handling problem.

2. Improved our edge-detection techniques on images to reduce the noise in the post edge-detected image. It was found that to find any lines within our detailed infrared images we needed to reduce the noise produced by the edge detection schemes that we had used to this point. We found that the Canny edge-detection scheme, along with our own second derivative analysis, proved to be very successful in reducing the noise associated with finding edges in the infrared image data that we had of tanks.

3. Created a filter to remove edge points not associated with the target of the image. Though the edge detector we now had worked very well, the result was the edge detection of the entire image. This was a problem because we did not want to find lines of objects that were not part of the target. We developed a filter that found the area of the target based on the detail of the edge detection results, and removed any excess from those results.

4. Created a mathematical solution to determine the lines formed from the resulting points of an edge detection filter. Since other line-finding techniques were both poor and fairly random, we developed a technique to determine exactly what lines are within a distribution of points. By creating a histogram of the slope and y-intercept of the

lines formed by joining every pair of points in the image that resulted from the edge detection, we could find the peaks of that histogram and therefore find the most prevalent lines in the point distribution.

5. Determined a method to reduce the lines to segments. Because the lines had no end points they would include isolated points or groups of points that were not associated with the main segment of each line. Many methods were tried and the result was an algorithm that allowed some small gaps in the lines, but not a large gap, nor a large number of small gaps. Gaps would occur because of the non-perfect distribution of points after the edge detection.

6. Applied these techniques to an image of a cube and to a CAD model of a cube. A cube was used to begin with because of the complicated nature of the tank images did not allow us to easily understand what was happening within our algorithms during early development. The cube provided an ideal environment to initially tune our algorithms, which we were successful at doing.

7. Applied these techniques to the images of a tank provided by the Army, and revised our techniques based on the problems that arose from these more complicated images. Moving from the cube to the tank demonstrated the difficulties with such detailed images. This was the impetus for developing improved edge detection schemes, a filter to remove edge detected points not associated with the main target of the image, and a method to reduce lines to segments, discussed in items 2, 3, and 5 of this section, respectively.

Section 3: Nearest-Neighbor Line Finding

A common technique for finding lines from a distribution of points, similar to that resulting from applying an edge detection mask to an image, is to start with a point, move to all of it's nearest neighbors. From there, determine if any lie along the line being created, and then do the same for each of the points included in that line. This technique, however, is always demonstrated using very few points, almost all of which fall within that line, and when looking for a line in a known direction. We were dealing with detailed images that, even with superb edge detection, create point distributions that are very complicated and full of indeterminate choices for the line finding algorithm. We also need to be able to find an undetermined number of lines that are in undetermined directions.

To start the algorithm we would pick a point in the scene, look at the surrounding square of eight pixels for other points. If none were found, we would expand the square to look at the next biggest square of fourteen pixels for other points. If other points were found, the line-finding algorithm would start off in the direction of the found points, looking of a continuous series of points in that direction. Before knowing whether a line exists in a certain direction, however, it is impossible to know how large gaps may be within a line or how far from the line points may stray. Giving up on directions because gaps were found at the starting point caused lines to not be found or to be cut-off short. Allowing for too large of gaps caused large numbers of "noise" lines to be found. Following points that strayed from a line would cause the algorithm to not know what direction a line should be continue. Not selecting these points when they were part of the line would reduce the number of points in the line and might cause the line to stray off course in the opposite direction.

A further problem to this method is that with undetermined lines, every point must be tried as the starting point for lines (a point may exist in multiple lines), causing the amount of time needed to complete such an algorithm with an image of any detail to become enormous.

We determined that this method of line finding would not be successful for our use.

Section 4: Improved Edge Detection

A better edge detector was needed to handle the type of data we needed to evaluate. The edge detectors that had been used previously in Phase I, namely Prewitt, Sobel, and Kirsch detectors, worked well with many color photos but had limitations with the infrared data, such as large amounts of noise and broken edge lines. These detectors tended to have too much noise and too little of the actual edges detected, or broken edges when the edges were found. After trials with other options, it was found that the combination of the Canny Edge Detector, discussed by Hancock, Kittler, and Petrou (References 1 and 2), and our second-derivative edge detector (described in our Phase I Final Report) was exceptional in comparison with other detectors.

The Canny edge detector takes the following steps:

1. The image data is smoothed by a two-dimensional Gaussian function of width specified by a user parameter. In practice, two-dimensional convolution with large Gaussians takes a long time, so that in practice we approximate this by two one dimensional Gaussians, one aligned with the x-axis, the other with the y axis. This produces two (rather than one) values at each pixel. The Gaussian function in one dimension is expressed as:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-x^2}{2\sigma^2}}.$$

2. Assuming two-dimensional convolution at stage 1, the smoothed image data is differentiated with respect to the x and y directions. It is possible to compute the gradient of the smooth surface of the convolved image function in any direction from the known gradient in any two directions.

Assuming the one-dimensional approximation at stage one, which we use, then the values in the x-smoothed image array are convolved with a first derivative of a one dimensional Gaussian of identical sigma aligned with y. Similarly, values in the y-smoothed image array are convolved with a first derivative of a one dimensional Gaussian of identical sigma aligned with x. Sigma is the user-set width of the Gaussian function. During the Phase I Option we used a sigma value of 1.0 for most operations. This width needs to be tested more extensively during Phase II to find an optimal level of performance.

From the computed x and y gradient values, the magnitude and angle of the slope can be calculated from the hypotenuse and arctangent.

The first derivative of the Gaussian function is expressed as:

$$G'(x) = \frac{-x}{\sqrt{2\pi}\sigma^3} e^{\frac{-x^2}{2\sigma^2}} .$$

3. Having found the rate of intensity change at each point in the image, edges must now be placed at the points of maxima, or rather non-maxima must be suppressed. A local maximum occurs at a peak in the gradient function, or alternatively where the derivative of the gradient function is set to zero. However, in this case we wish to suppress non-maxima perpendicular to the edge direction, rather than parallel to (along) the edge direction, since we expect continuity of edge strength along an extended contour. The second derivative of the Gaussian function is expressed as:

$$G''(x) = \frac{-1}{\sqrt{2\pi}\sigma^3} e^{\frac{-x^2}{2\sigma^2}} \left[1 - \frac{x^2}{\sigma^2}\right].$$

Rather than perform an explicit differentiation perpendicular to each edge, another approximation is often used. Each pixel in turn forms the center of a nine-pixel neighborhood. By interpolation of the surrounding discrete grid values, the gradient magnitudes are calculated at the neighborhood boundary in both directions perpendicular to the center pixel, as shown in Figure 4.1, below. If the pixel under consideration is not greater than these two values (i.e. non-maximum), it is suppressed.

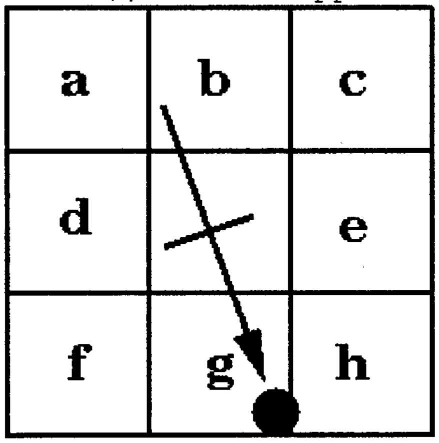


Figure 4.1: From central gradient value, interpolate gradient value at dot from gradient values at e, g, and h. Repeat in opposite direction. Suppress if non-maximum.

4. The thresholder used in the Canny operator uses a method called "hysteresis". Most thresholders used a single threshold limit, which means if the edge values fluctuate above and below this value the line will appear broken (commonly referred to as ``streaking''). Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if a value lies above the upper threshold limit it is immediately accepted. If the value lies below the low threshold it is immediately rejected. Points which lie between the two limits are accepted if they are connected to pixels which exhibit strong response. The likelihood of streaking is reduced drastically since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur.

Figures 4-2, 4-3, and 4-4 show the results of thresholding the same tank image with Prewitt, Sobel, and Canny edge detectors, respectively. Notice the improved signal-to-noise and the reduced line-breaking in the Canny threshold image.

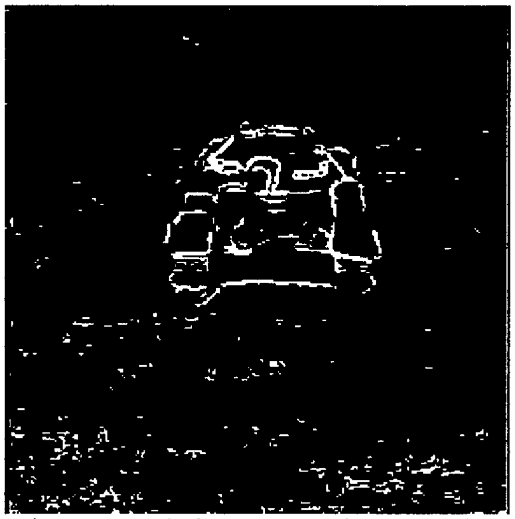


Figure 4-2: Prewitt edge detection of tank image.

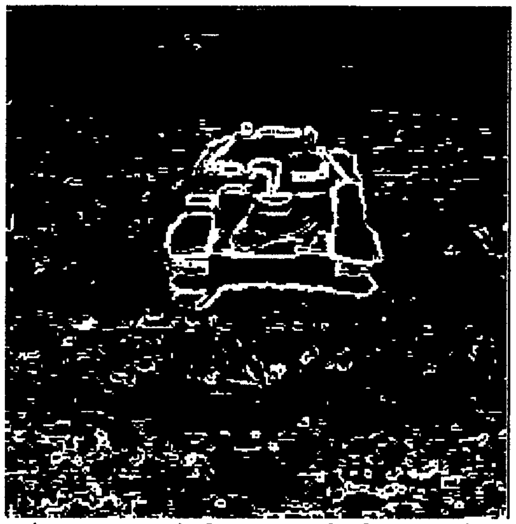


Figure 4-3: Sobel edge detection of tank image.

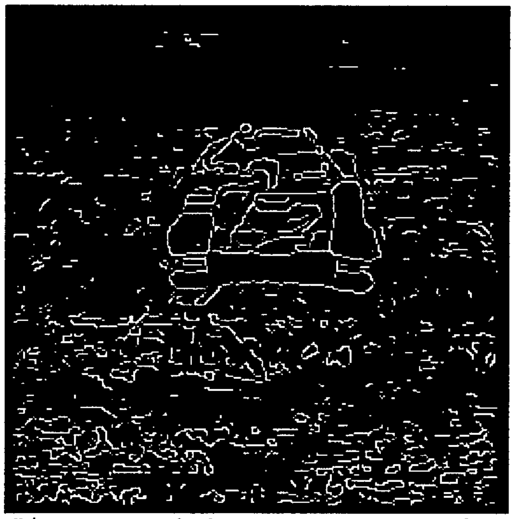


Figure 4-4: Canny edge detection of tank image.

Section 5: Filtering Non-Target Data

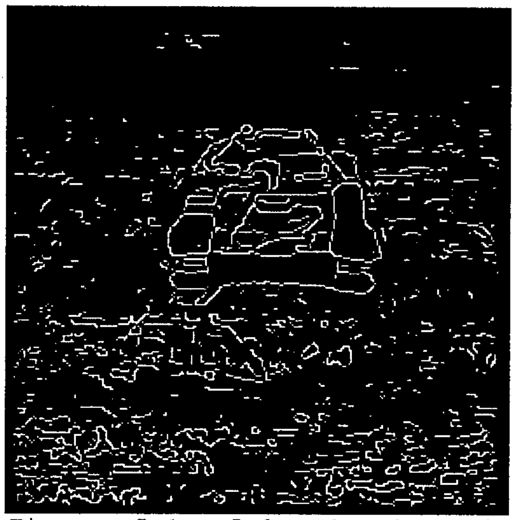
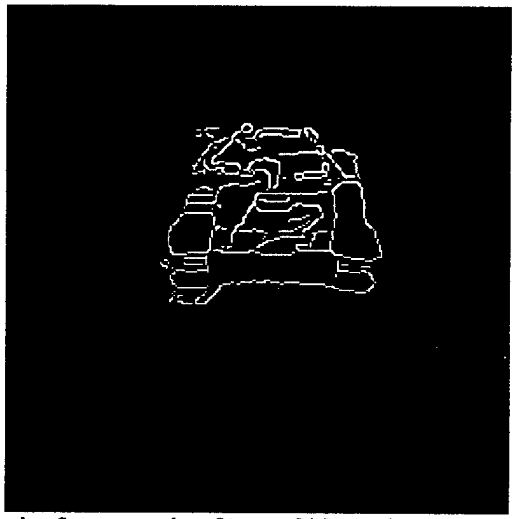
After using the new edge detection scheme, though most of the noise was cleared from the resulting point distribution, there remained the problem of edges detected that were not associated with the target of interest. To solve this problem a filtering algorithm was developed that removed the extraneous points from the resulting image.

This filter is based on the assumption that the important points of the target are contained within long strings of connected points. We call them strings and not curves or lines since we allow the branching off of curves at every point, just as long as we have no space between points. With the infrared data that we have in our possession, this seems to be an accurate assumption.

A list of strings is created by starting at a point, assigning it to a new string, searching all surrounding pixels to that point for other points, and adding any bordering points to the same string. Then for every found point, proceed with the same logic, until all points are assigned to a string.

The largest string is then found, and all points belonging to strings that are within a certain percentage of the largest string are kept, while all other points are discarded. This percentage is currently set manually and we have found that 70% of the largest string works well with our specific data. We hope to find a method of automatically choosing what size of string to keep.

Figures 5-1 and 5-2 show the result of an edge detection scheme before and after the use of this filter.



Figures 5-1 & 5-2: Edge detection before and after filtering.

Section 6: Lines from Histogram

Because other methods of finding lines proved to be inadequate, and relied on user selection at that, we sought to find another way to identify lines within the distribution of points resulting from the edge detection and filtering algorithms. A purely mathematical solution was sought, and was found.

The new method involved creating a histogram that counted the number of lines created by each pair of points in the image. A 2-dimensional histogram was created, where one axis was the y-intercept and the other the slope (A variation on this definition of a line was used because both slope and y-intercept continue to infinity and do not lend themselves to creating cells in a histogram. The variation involved finding the angle the line made with the horizontal, from -pi to pi radians, and the closest distance of the line to the point in the center of the image. The distance was considered to have direction so that a line on one side of the center would not be confused with a line on the opposite side of the center with the same slope or angle and distance magnitude).

Figure 6-1 shows an original cube image, figure 6-2 shows the cube after edge detection and filtering was performed, and figure 6-3 shows the results of creating such a histogram from that cube. Nine peaks in the histogram appear far above all other noise in the histogram, and the locations in the histogram match the slope and y-intercept of the obvious lines in figure 6-1.

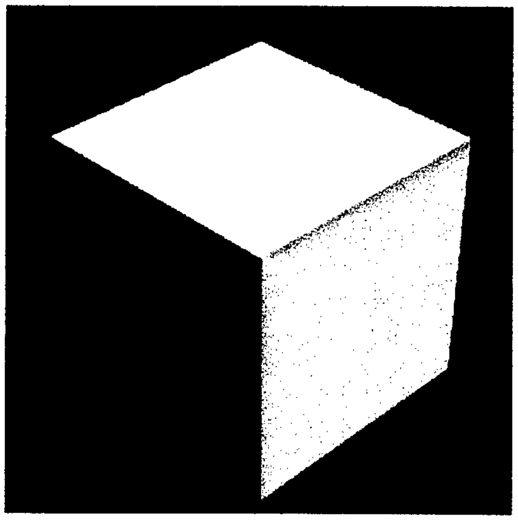


Figure 6-1: Image of cube used during Phase I Option testing.

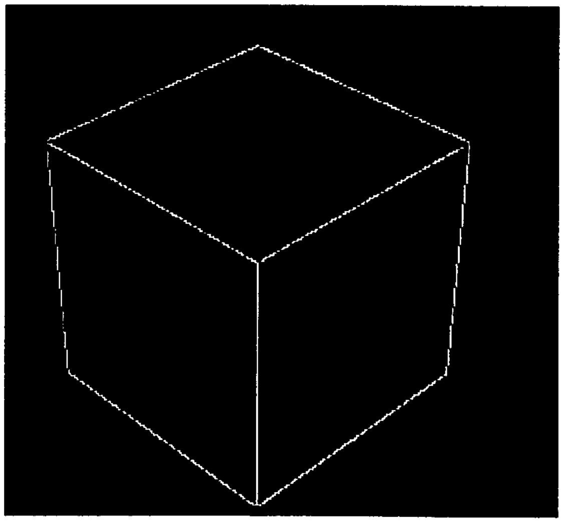


Figure 6-2: The result of using a Canny edge detector and target filter on a cube.

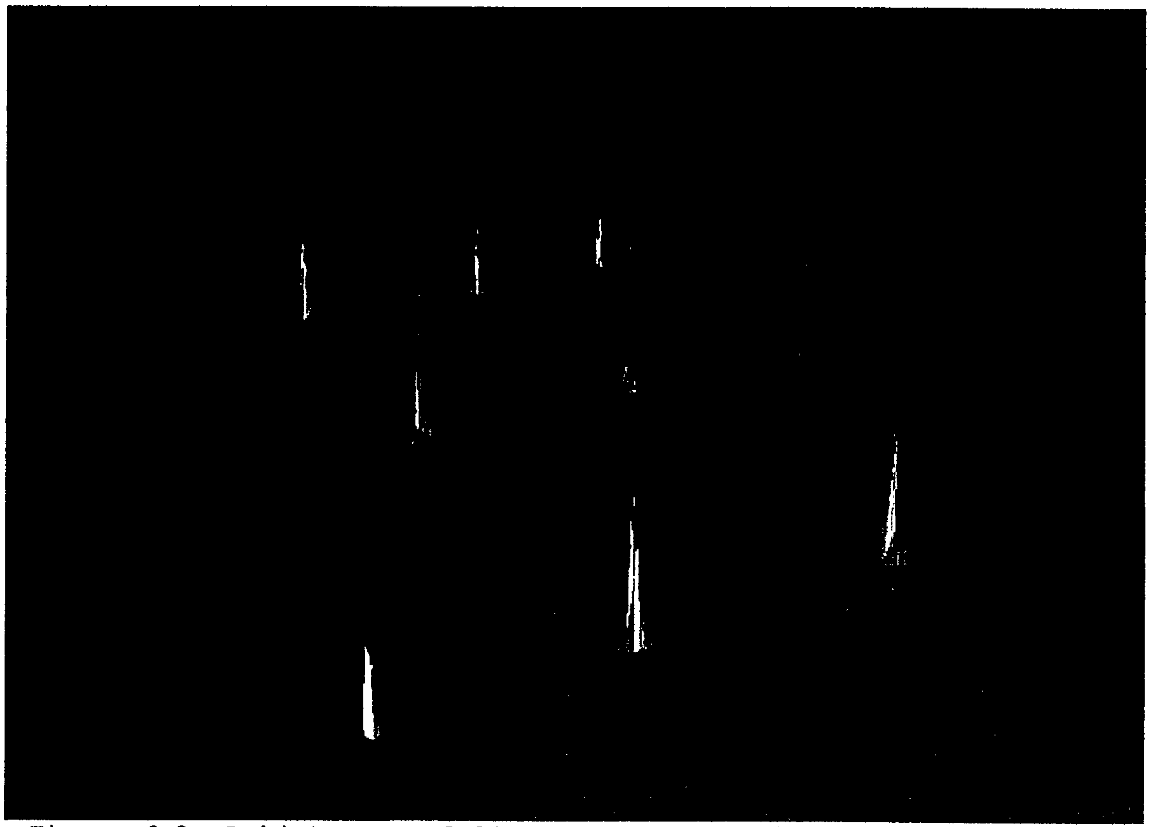


Figure 6-3: A histogram of lines formed by point pairs for a cube. Angle of line runs approximately from left to right (and wraps), and distance from center from a negative distance (far end) to a positive distance (near end). In this figure, the tallest peak is blue, then purple, then yellow, and those in the noise are green.

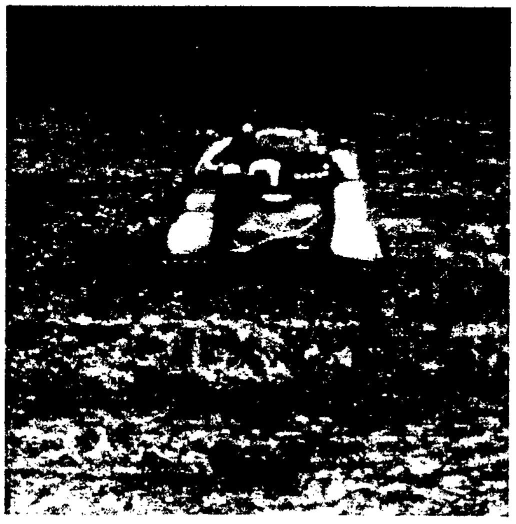


Figure 6-4: One of several images of a tank used during Phase I Option testing.

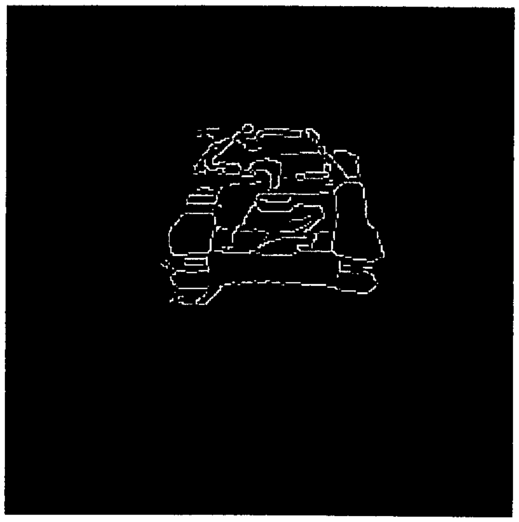


Figure 6-5: The result of using a Canny edge detector and target filter on a tank image.

Figures 6-4, 6-5, and 6-6 show the same examples for an image of a tank. Notice that straight lines are not as prominent, and that the peaks do not stick out of the noise nearly as far.

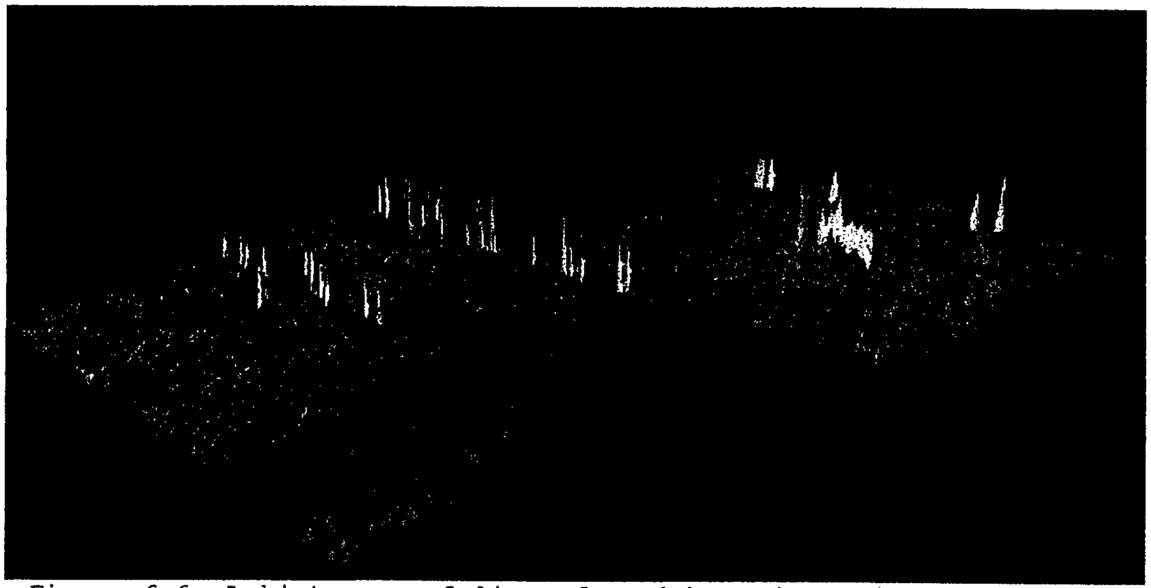


Figure 6-6: A histogram of lines formed by point pairs for a tank. Angle of line runs approximately from left to right (and wraps), and distance from center from a negative distance (far end) to a positive distance (near end). There are many horizontal lines from this image, as can be evidenced by the many peaks down the middle (at angle = 0 degrees). The peaks are color-coded by height, with the green peaks in the noise.

To recreate the lines from the histogram, the position of each peak was found in the histogram. Then, all points in the post-edge detection image that lay upon the lines defined by the slope and y-intercept of these peaks would be added to a list of points contained in that line.

When moving to more complicated images, many adaptations were necessary to correct for some of the imperfections of discrete mathematics of pixel geometry. Commonly the slope of the line from the peak may be off by significant fractions of a degree, as well as the distances being off by fractions of a pixel.

A weighted-smoothing function was used when defining the locations of the peaks. After peaks were found by finding 2-dimensional local maxima among the locations' neighborhoods, and peaks within the noise were eliminated, a situation existed with clusters of double or triple peaks at many locations. A weighted-sum of the cluster was used to combine the cluster into a single peak. This also appeared to cause the peak to agree more with the point distribution, causing more points to lie along the line defined by the peak.

Since each line is actually infinitely thin, and because of some slight errors caused by the discreteness of the image, it was necessary to allow some deviation from the straight line defined by each peak when deciding which points belonged to each line. Therefore, points that lay within a small perpendicular distance to the line were included.

Section 7: Lines To Segments

Lines were originally sought so that we could find the intersection of lines, which would relate to corners on the surface of the target. The lines themselves, then, relate to actual edges of the target.

The set of lines found by using the histogram method described in the previous section were drawn on the screen by using the points from the post edge detection image that fell on these lines. Though the proper line would be drawn, many points that existed far away from the main segment were not part of the same edge, and generally were not part of an edge that was oriented in the direction of that line. In addition, and more importantly, the fact that lines are infinite (or at least extend to the edge of the image in this case), many lines intersect at points not associated with corners of the target as they are not along edges of the target at the point of intersection.

One might try to check whether the intersection occurred at a location where points existed on both lines. Two problems exist with this solution: one being the aforementioned situation where points exist on the line far beyond the end of an edge, and the other being that gaps exist in the list of points that are along the edge due to the imperfections of edge detection.

So a method was required to find end points to the lines, changing the lines to line segments.

Many ideas were experimented with, but the method we have settled on is an algorithm that allows some gaps in the points, but not too many or too large of gaps.

The algorithm starts by looking for the longest continuous section of the line. A continuous section is defined as a line segment with no space between consecutive points. A continuous section may contain one or more points. All of the continuous sections are found, and the section with the most points is considered the base of the ultimate segment to be found.

Moving both directions along the line we add at most two more continuous sections each direction. Requirements for incorporating the additional sections are that there is no more than a certain maximum distance (gap) between sections, and the cumulative gap is not larger than a certain value. Both the maximum gap size and the maximum cumulative gap are user-defined parameters, as is the number of continuous sections incorporated into the segment (though we found that two on each side definitely tends to be as many as is wanted in the majority of cases). The result was a collection of line segments that defined some of the edges of the target being observed.

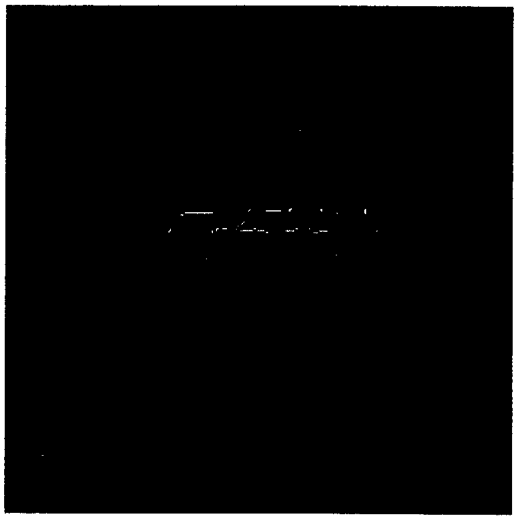


Figure 7-1: Lines derived from a tank image, using the histogram method of Section 6.

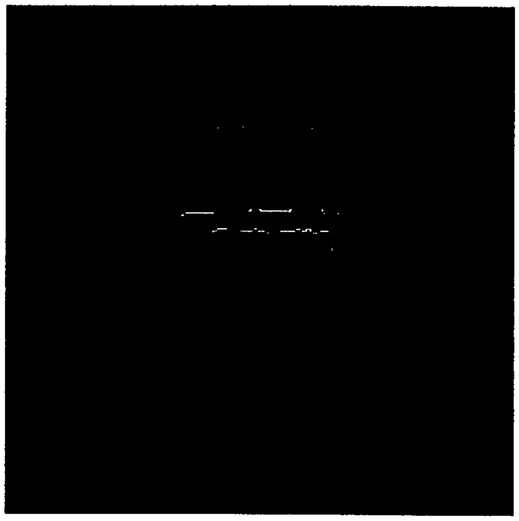


Figure 7-2: Segments extracted from lines using the segment-finding algorithm.

Figures 7-1 and 7-2 show the lines and segments, respectively, derived from an image of a tank. The algorithm currently removes too much information in terms of number and length of segments, reducing the number of intersections among the segments (which will be discussed in the next section). This algorithm will be improved in Phase II of this contract.

Section 8: Finding Intersections

We are interested in finding features of the target that will allow us to compare it to CAD models to find the quality of a match. Though edges are certainly features, they provide little in the way of information that will allow us to align the images with CAD models; they are 1-dimensional in a 2-dimensional medium. Intersections of these edges, however, will form a 2-dimensional feature in these images that will provide greater amounts of information for image alignment.

Once the edges of the target are found in the form of line segments, the edges can be found by simple line-intersection algorithms. For example, we will use the algorithm:

Let L1 and L2 be two line segments we are testing for intersection, and let (x1,y1) and (x2,y2) be the end points of L1 and (x3,y3) and (x4,y4) be the end points of L2.
Set $d1x = x2 - x1$, $d1y = y2 - y1$,
$d2x = x4 - x3$, $d2y = y4 - y3$,
then the vectors defining the lines L1 and L2 are <d1x,d1y> and <d2x,d2y> so that the equation of segment L1 can be written:
$$L1 = (x1,y1) + t * \langle d1x,d1y\rangle,$$
where $0 <= t <= 1$, and similarly for L2. Since we know that two vectors are parallel if their cross-product, L1 x L2, is equal to zero, we set
$$delta = L1 \times L2 = d1x * d2y - d1y * d2x.$$

If delta = 0 then there is no intersection, otherwise the infinite lines (not necessarily the segments) do intersect. To find the point of intersection remember that:
$$L1 = (x1,y1) + t * \langle d1x,d1y\rangle \text{ and}$$
$$L2 = (x3,y3) + s * \langle d2x,d2y\rangle.$$

The point of intersection, in terms of parameters t and s will be:

```
     t = ( (x3 - x1) * d2y - (y3 - y1) * d2x ) / delta,
and  s = ( (x3 - x1) * d1y - (y3 - y1) * d1x ) / delta.
```

This point of intersection is within the segment if and only if t and s are both between 0 and 1.

To find the point (x,y) of intersection, simply plug into either line equation. Using L1, we find:

```
     x = x1 + t * d1x, and
     y = y1 + t * d1y.
```

To categorize the intersection we again look at s and t. Many of the intersections tend to be corners where the segments do not extend beyond the intersection. In this case, t will be either 0 or 1, as will s. If just one of these two parameters is either equal to 0 or 1, and the other resides somewhere between 0 and 1, then we have a T intersection. If both parameters reside between 0 and 1, then we have a cross. These are the three types of intersections that we will use in Phase II to align and match infrared images with CAD model images.

Section 9: Cube Applications

To develop and test many of our ideas, a simple CAD model and an equally simple and clear image with little detail other than the specific target was used to begin with. For the development of the line algorithms a cube was chosen due to its simplicity and the existence of many corner features.

First a CAD model of a cube was created. Then, an observer position and viewing direction relative to the cube's CAD model coordinates that would allow a view of three sides of the cube was chosen. The CAD model was rendered with this view, and an image of that rendering was saved. This became the representation of the infrared image that was tested.

Originally the Nearest-Neighbor line finding algorithm was tested on a more complicated image, but then on the cube. Even on the cube, very poor results were obtained with this algorithm. Then the Histogram Line Finding algorithm was tested on the cube, with excellent results. This gave us confidence that the histogram method was indeed both accurate and very useful. It became apparent, however, while testing with the cube, that the lines would have to be shortened to segments. Infinite lines would not only discover the points associated with the edge of the cube being found, but also with a point or two on a different edge, often quite far away from the original edge.

Upon successful experimentation with the cube, the histogram line-finding method was deemed appropriate for use with the ARTUS software. It was time to test these methods with a more complicated model, but the cube will be used in future testing as well.

Section 10: Tank Applications

To test our software applications fully, data similar to that which would be used in the field, was used so that the algorithms' effectiveness against real data could be measured. Infrared images of a tank were used. While using the tank image, algorithms that were successful against the cube image proved to be far less effective against the tank image.

Though the histogram-based line-finding algorithm definitely turned up lines from the tank, it also turned up a large amount of noise; so much noise that it was difficult to make out the lines from the tank. By examining the histogram closely it was determined that the tank lines were the most prominent, but the noise levels were so high that it was difficult to make a distinction between the two. Somehow the noise had to be reduced.

The first step taken to reduce noise was to change the edge-detection algorithm. We had formerly used a Sobel, Prewitt, or Kirsch edge detector, adding our own second derivative detector on the tail end (all of these are described in the Phase I Final Report). We kept the second derivative tail end, but changed the front end to a Canny edge detector, described in Section 4. The Canny/second derivative edge detector was ideal for the infrared data that we were studying, eliminating almost all of the noise.

Though much of the noise was removed, there remained many lines not associated with the target since there really were edges of objects not associated with the target within the image. A way to determine which lines were part of the target and which were not was needed, so the algorithm of Section 5 was developed.

Section 11: Conclusions

The Phase I Option portion of this contract developed one part of the ARTUS system, transitioning from the planning stage of Phase I to the system development stage of Phase II. After examining different methods, we were successful and creating a method to extract lines from infrared images. The method involved creating a histogram of all of the possible lines within the results from an edge-detection process. This operation required improved edge-detection from our earlier work, as well as a new algorithm to filter the target data from the non-target data. Both of these tasks were successfully completed.

Following the location of lines, we created an algorithm to parse the lines into line segments. This process was partially successful, but needs further enhancement during Phase II to allow for more line segment intersections. All of the line intersections that were possible to find were successfully located using a newly developed algorithm to find line segment intersections.

During Phase II this portion of the ARTUS system will be incorporated with the other necessary components of the system, such as image alignment with CAD models, match quality measurement, difference detection, and rapid CAD model updating.

References

Hancock, E.R, and Kittler, J, Adaptive Estimation of Hysteresis Thresholds, Computer Vision Processing 1991 (196-201).

Petrou, M, Separable 2-D Filters for the Detection of Ramp Edges, IEE Proceedings - Vision, Image, and Signal Processing, No.4, 1995.